BULLETIN OF THE POLISH ACADEMY OF SCIENCES
TECHNICAL SCIENCES, Vol. 68, No. 6, 2020
DOI: 10.24425/bpasts.2020.135395

Texture recognition system based on the Deep Neural Network

R. KAPELA*

Poznan University of Technology, ul. Piotrowo 3A, 60-965 Poznan, Poland

Abstract. This paper presents a deep learning-based image texture recognition system. The methodology taken in this solution is formed in a bottom-up manner. It means we swipe a moving window through the image in order to categorize if a given region belongs to one of the classes seen in the training process. This categorization is done based on the Deep Neural Network (DNN) of fixed architecture. The training process is fully automated regarding the training data preparation, investigation of the best training algorithm, and its hyper-parameters. The only human input to the system is the definition of the categories for further recognition and generation of the samples (region markings) in the external application chosen by the user. The system is tested on road surface images where its task is to categorize image regions to a different road category (e.g. curb, road surface damage, etc.) and is featured with 90% and above accuracy.

Key words: deep learning, texture segmentation, artificial intelligence.

1. Introduction

Texture recognition has always been a challenging task among the image processing problems. On the other hand, texture brings a lot of information to the categorization of the object in the images. This is the main reason for the investigation of the algorithms of texture categorization and texture-based object segmentation. Over the years, different approaches to the problem have been used. Due to the relatively long history of the subject and its diversity, only a few exemplary leading papers will be presented.

Early approaches to the problem were characterized by the frequency analysis of the image function [1]. For this purpose, some type of transition between image and frequency domains were used. To the most popular approaches, the fast Fourier transform and its derivatives can be included. Next, the frequency clustering was usually used on the signal received from the previous stage. Usually, a spectrum of higher frequencies was taken with higher precision into consideration for texture categorization. For this purpose Gabor filtering was mostly used. A feature vector was composed of a grouped by a set of Gabor filters frequencies calculated based on the texture of the region [2]. These algorithms were relatively simple so that their hardware realizations were feasible [3,4]. With today's technology trends, texture analysis is relatively complex and calculations of these algorithms are rather done in GPUs with the use of CUDA technology if more computational power is needed. What has to be mentioned at the beginning of the paper is that the algorithms and solutions presented herein do not serve the task of texture mapping [5] since this is a very different task. Also the main emphasis of this work is not placed on the medical imaging texture recognition. For this reason only papers from this field will not be mentioned further on and compared in experiments and conclusion.

Historically there has been an enormous amount of research on texture processing. Since, as was mentioned, in the early 2000s usage of GPU was rather not popular because of its cost the mainstream of research in texture recognition was based on fast and accurate texture feature extraction. There has been a significant amount of work in that area. Thorsten et al. [6] for example analyze texture features based on the Human Visual System hypothesis. The most common features form a set of descriptors called a visual texture property of a region and it seems to be domain-independent. The system is however slow and inaccurate in non-rich texture regions. A similar approach is presented in [7] where the authors organize the texture features in a vector of key-points that allows the forming of a descriptor for further recognition based on a very well known technique called bag-of-features. This is convenient since it does not require a big training set due to the fact that texture descriptors can be easily clustered. A more robust algorithm of the texture classification has been used in Guo et al. work [8] where some invariance to rotation, image illumination, and noise is achieved. This is done by using the Fisher Separation Criteria in a learning framework of texture descriptors. This way the texture features are guaranteed to have optimal distance among different classes. As a drawback of the solution, one can mention that the system can only be fed with images that present a single texture, thus, reducing the number of possible usage scenarios.

Recent works most often use DNNs as a universal feature of extractor and classifier. These solutions are featured with very high accuracy and robustness that outnumbers earlier approaches. However, they differ with the approach taken in order to solve the given problem. This often results in different system architecture. Comparable works to the application pre-

*e-mail: rafal.kapela@put.poznan.pl

Manuscript submitted 2020-03-27, revised 2020-08-10, initially accepted for publication 2020-09-02, published in December 2020

sented here can be found in [9–11] where focus of the first two is on the effective feature extraction for further accurate classification. This way in [9] we can observe the usage of LBP again as a pre-processing technique but this time inside a deep architecture of a neural network. Two major deep architectures were investigated: late and early fusion where texture features are being fed to the system at different stages of processing. This is very interesting since it seems a proper texture feature handling is the key to the high accuracy and robust recognition. In [10] focus is put also on the texture features but this time they are hand-crafted and the rest of the architecture is re-trained to correctly classify them. Both systems are featured with high recognition accuracy but they lack the ability to recognize multiple texture classes in the same image. Zhu et al. [11] on the other hand deal with the problem of data augmentation which is very common in training the models with tens of millions of parameters. The system described in this work has very similar advantages and disadvantages.

The author's reasoning was to design and build a texture recognition system that is featured with high accuracy and robustness presented in recent papers that is also capable of recognizing multiple regions in the image as the separate objects. In addition, the system is designed in a way that allows fully automated data augmentation and a parallel investigation of efficient model training parameters.

The remainder of the paper is organized as follows: Section 2 provides information about the system architecture. More specifically data pre-processing followed by the description of the data acquisition for training and recognition processes are presented. Next, the deep learning module and its training procedures are presented. Paper ends with Sections 3 and 4 where system recognition results and conclusions are shown.

2. System description

The input to the system is the gray-scale image of any resolution. The texture recognition algorithm does not re-scale input but it works by dividing the input image into non-overlapping segments and then do the processing within each segment. This is true for training the model as well as further on-line recognition. Note, that due to this approach the origin of the image (e.g. its recording/capturing parameters like exposition, resolution, etc.) are neglected and it is up to the user of the system to prepare sufficient quality images. In the asphalt dataset presented herein images were recorded by an automatic image data capturing vehicle [15] designed especially for this task but other datasets presented herein are public domain and the procedure of data capturing process is unknown in these cases. The block diagram of the systems architecture is presented in Fig. 1.

As can be seen, the system is capable of working in the two modes: train mode and recognition mode. While the mode names are self-explanatory, the way they work is much more difficult. Both of them share the same pre-processing track that is responsible for forming the input signal (image) to the rest of the processing pipeline. Then, depending on the mode the way the system works to change. Both modes are explained below in more detail.

2.1. Data pre-processing and segment formation. Data pre-processing segment is shared between the two modes of the application. The way it operates is the following:

- calculate the low/high levels of the pixel values (5% of cumulative histogram starting from 0(up)/255(down) respectively);
- applies the histogram leveling accordingly to the detected levels;

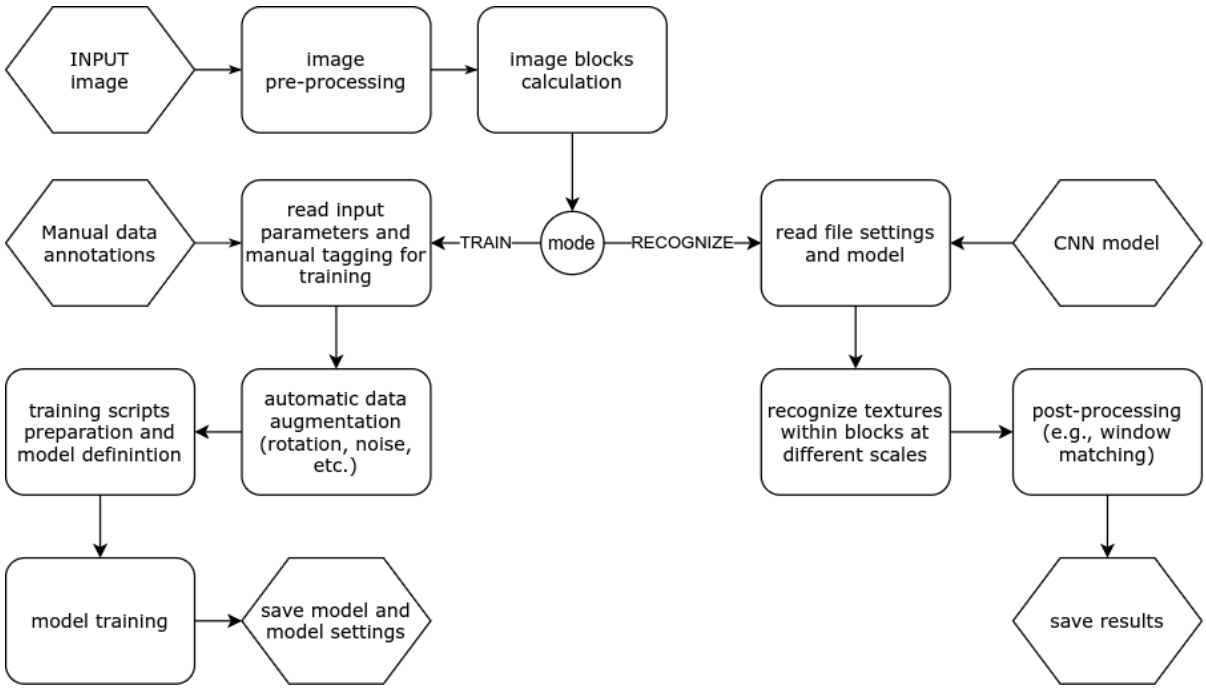
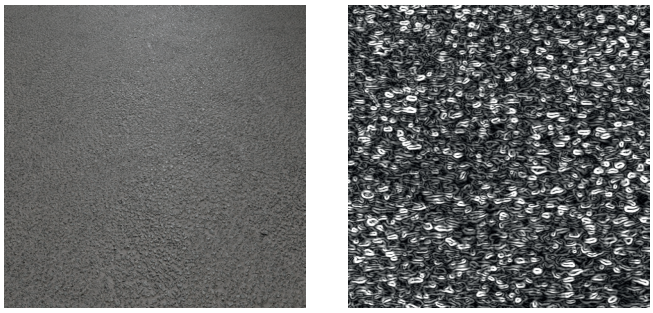


Fig. 1. Block diagram of the system

- removes the noise from the image by applying the Gaussian filter (by default the Gaussian filter mask is 5×5 pixels with $\sigma = 0.83$);
- detects the edges with Laplace edge detector.

Figure 2 shows the exemplary input image and its pre-processed version with all the above steps applied.



(a) (b)

Fig. 2. Input pre-processing: (a) original image; (b) pre-processed image

The next step just after the pre-processing is to form image segments accordingly to the setup parameters. Since the main processing is based on the image segments system is designed in a way that allows adjusting the size of it. The default size of the segment is 25×25 pixels. In addition, the system allows us to recognize textures at different scales, thus, the image pyramid is built as well. For this reason the window multiplier parameter is used. The idea of block size and window multiplier is shown in Fig. 3. Recognition mode also allows us to mark the regions that are composed of multiple segments. This way, there is no need to calculate overlapping segments in this procedure. For this purpose, the system accepts an additional parameter that tells how many positive segment recognitions should be inside one window to mark the region as positive.

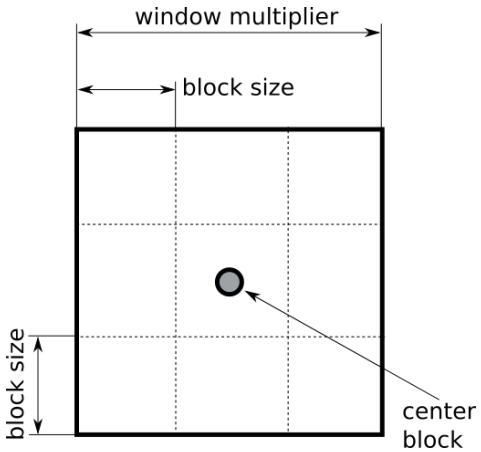


Fig. 3. Input split idea to segments and windows

In the training mode of the system extracted segments are labeled accordingly to the manual annotations given in the auxiliary file. This way each element gets a class label that it belongs to. Based on that system can detect automatically how many classes must have the model on the output. Obviously, the number of classes is $N+1$ due to the fact that a "negative" class is extracted automatically from the samples that do not belong to any manual annotation. The information about the number of classes goes to the model definition segment where the model is constructed.

2.2. Data augmentation. The number of training samples in the presented system depends on the input image resolution (since input images are divided into non-overlapping segments). Since the size of the window is 25×25 pixels the number of samples from one input image is: $floor\left(\frac{W}{25}\right) \cdot floor\left(\frac{H}{25}\right)$. In order to train a deep neural network effectively typically, several tens of thousands of training samples are needed. This is a requirement since the model usually has about 50 thousands of parameters or more. Small training set during training results with great train accuracy and perfect confusion matrix but the model will not work correctly on the data it has not seen before. Model overfitting with small training sets happens even if the validation set is defined. To avoid that data augmentation is used. The way the data can be augmented depends on a specific scenario. The system presented herein can be adapted from this perspective without changing its architecture. To achieve this a separate batch script can be invoked on the data. Its purpose is to create more training data so it is transparent to the rest of the system. The system implements two default augmentation techniques: displacement and rotation. Both prepared as bash scripts for the Linux operating system.

The displacement script prepares a set of a few images around the original sample. The number of these additional samples can vary between zero and eight (default is eight). If the segment size is the same as the input image it cannot be used since there is no padding available.

The rotation module is also controlled with a system parameter that gives information about the maximal image rotation. Again, the image ImageMagick library was used. By default system also adds the horizontal and vertical mirror effects since they do not cause any aliasing (operations `-flip` and `-flop` in the library).

2.3. Deep learning module – model definition. Having all the necessary information from the previous stages system automatically creates a deep neural network model input and output layers. Numbers of layers in the model as well as their sizes and parameters are defined as a separate sub-model file and can be redefined there if there is a need for it. The system adapts the input and output layers so that they match the dimensionality of the input/output signals. The backbone model used as a sub-model uses four convolution layers followed by the pooling and normalization layers. The input layer accepts images of resolution equal to 256×256 pixels (automatic resizing is done for other dimensionality of input images). The number of convolutional kernels in the following

layers is respectively: 32, 64, 64, 128. The last two layers are composed of fully connected neurons with ReLu activation functions [13]. Their sizes are 200 and $N+1$ respectively. The number of layers and parameters in each layer was adjusted based on experiments so architecture presented herein is a resultant of all the experiments performed on this system (e.g. in the asphalt dataset recognition experiment number of parameters in the entire model is 49872). Note, that it can be adjusted in a separate sub-model file for the more specific cases. This architecture is the result of many experiments and is a default setup of the system. Since each time the model is trained its weights are randomly chosen and there is no guarantees in the training algorithm to reach a global minimum it is impossible to say that this choice is optimal. However, based on the experiments performed it is quite often a good choice.

Due to its popularity and ease of use the Caffe library was used in the project [12]. Presented DNN model definition in the Caffe library uses *Xavier* algorithm as a weight initializer [14]. The same technique is used in all the convolution layers. The model definition is stored in a separate file for further training. Training parameters are stored in an additional file. The DL backbone model definition as well as the default training setup for caffe library can be downloaded as a prototxt definition files from http://rafal.kapela.pracownik.put.poznan.pl/demos.html#.

2.4. Deep learning module – model usage (training and recognition). The defined model, depending on the system mode can be trained or used for recognition. In training mode, different setups can be used as it was mentioned in Section 2.3. Exemplary visualization of training processes is presented in Fig. 4. Separate files for model and solver definitions allow us to run the training process in the batch mode on a grid of computers. This is very convenient for the exploration of the parameter space for the given task of texture recognition.

Once the model is trained it is saved at a given location. Then, it can be used in recognition mode. Caffe library allows to evaluate the model with built-in tools as well. This is done after the training on the unseen images – not the ones from the validation dataset. This allows us to quickly and efficiently check if the results are satisfactory for further recognition.

3. Experimental results

The experimental results presented herein are based on the asphalt image dataset presented in one of the author's previous work [15]. It consists of 3000 gray-scale images, 1500×1080 resolution each. The algorithms presented there have nothing in common with the automatic texture categorization system described in this work. Note, that there are multiple, open texture datasets that are suitable for this system (e.g. [16]). Results for some of them are also presented herein. To train the model a ground truth labeled dataset is needed in each case. It is usually done in two ways – either each image in the database presents a separate texture (then we do not need image splitting into the segments) and is assigned to a particular label or images are given in a bigger resolution and additionally, there is an auxiliary file with a description of the image segments.

The asphalt dataset used in this research consists of multiple texture classes but for our purposes only six of them were chosen for system accuracy validation. These were:

1) regular asphalt surface;
2) asphalt crack;
3) curb;
4) gutters;
5) surface contamination;
6) hatch sewer manhole.

The detailed results of the influence on the accuracy of each class have been presented herein. The main purpose of these results is to show the system capabilities, mainly that this is a scalable and flexible application that can be easily run in the batch mode on the cloud servers. The experiments were conducted for different parameters of the network solver. When one of the parameters was changed the rest were untouched in order not to show the influence of the correlation of the parameters. The default values for all the experiments are as follows:

- training algorithm – Adadelta;
- learning rate (η) – 0.001;
- learning rate modifier ratio (γ) – 1;
- momentum (μ) – 0.9;
- weight decay (λ) – 0.005.

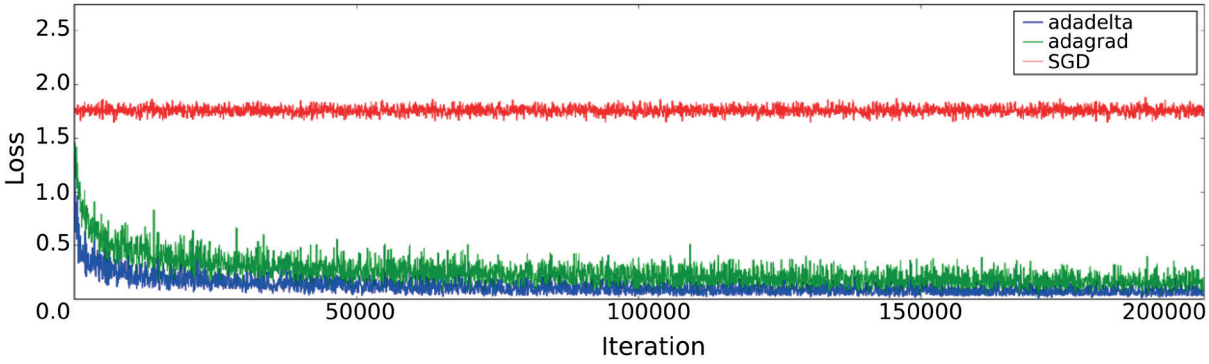


Fig. 4. Visualization of the training process

3.1. Training algorithm. Different training algorithms have been tested in order to show the capabilities of the system. It obviously depends on the task but what can be observed in general is that the adaptive gradient DNN training methods perform better than standard gradient descent methods (Fig. 4) [17]. This is due to the fact that the update of the parameters during the training is done separately for each one of them (1).

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\varepsilon I + diag(G_t)}} \cdot g_t \,, \tag{1}$$

where η is the initial learning rate, ε is a scalar, I is the identity matrix and g_t is the gradient of the loss function L calculated at a given step t (2).

$$g_t = \frac{1}{n} \sum_{i=1}^{n} \left(\nabla_\theta L(x^i, y^i, \theta_t) \right). \tag{2}$$

The loss function used in all the experiments is a well-known hinge (SVM) loss given by Eq. (3).

$$G_t = \sum_{\tau=1}^{t} g_\tau g_\tau^T \,, \tag{3}$$

where s denote a scalar score value observed at the output of the network.

The key of this algorithm is a diagonal matrix G which stores the accumulated gradient during the entire training process (4).

$$L_i = \sum_{i \neq y_i} \max(0, s_j - s_{y_i} + 1). \tag{4}$$

Adadelta training algorithm is an improvement of the adagrad procedure. It simply replaces the diagonal matrix $diag(G_t)$ with the decaying average over past squared gradients (5).

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\varepsilon I + E\left[diag(G_t)\right]}} \cdot g_t \,. \tag{5}$$

This way the denominator in (5 becomes a root mean squared (RMS) error criterion of the gradient. In the task of texture recognition and, in general, recognition of complex patterns adaptive subgradient training algorithms declassify basic gradient descent algorithms. The difference is extremely significant – over 70% increase of accuracy for the adaptive subgradient methods.

3.2. Learning rate. Table 1 shows the calculated by the presented system influence of the learning rate (η) parameter on the final results of texture classification. As it was shown in the previous section adadelta algorithm provides the best results so that it was used as a default training algorithm for all the experiments. For simple and distinctive textures (class 6) the influence of this parameter is barely visible and almost any value from the wide range (10^{-5} to 1) can be safely taken. For more complex textures there is a clearly visible extremum of the accuracy function. For this reason, a learning rate value investigation should be always taken into account when new models are created. The presented system can do that in a fully automated manner.

Table 1
Learning rate parameter influence on tested class accuracy in the system

class	η					
	10^{-5}	10^{-4}	10^{-3}	10^{-2}	10^{-1}	1
1	0.294	0.591	0.719	0.914	0.936	0.751
2	0.062	0.565	0.699	0.995	0.964	0.882
3	0.886	0.992	0.991	0.999	0.997	0.998
4	0.693	0.983	0.998	0.976	0.996	0.988
5	0.472	0.799	0.851	0.96	0.995	0.792
6	0.843	0.998	0.997	0.999	0.994	0.999

3.3. Learning rate modifier ratio. The second very important parameter is the learning rate modifying ratio (γ). Its importance has been shown in Fig. 5. As can be seen, some values of the parameter can lead to an inability to move out from the plateau in the loss function during training.

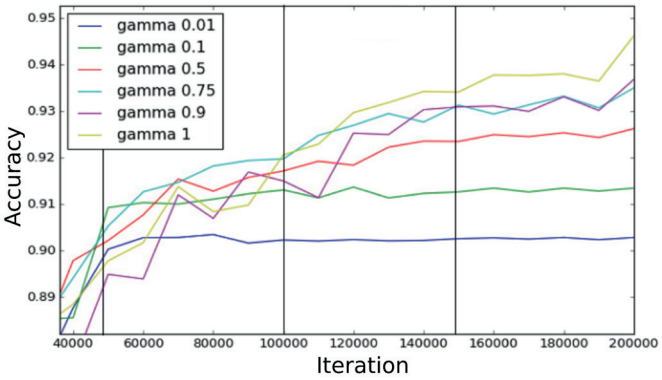


Fig. 5. Influence of the gamma parameter on the training accuracy

Table 2 shows the final influence on each trained class. Again, for trivial textures that are easily discriminative in terms of texture features the value of this parameter does not show too

Table 2
Gamma parameter influence on tested class accuracy in the system

class	γ					
	0.01	0.1	0.5	0.75	0.9	1
1	0.619	0.683	0.641	0.653	0.651	0.732
2	0.193	0.389	0.359	0.471	0.443	0.669
3	0.985	0.977	0.987	0.993	0.991	0.991
4	0.923	0.933	0.97	0.983	0.984	0.988
5	0.793	0.785	0.849	0.748	0.825	0.851
6	0.998	0.995	0.999	0.999	0.999	0.997

much importance since the entire tested range is equally effective. However, for more complex textures that are close to each other in the feature space a swipe through a range of different values is extremely useful for good accuracy results. Note, that the table shows the results for γ parameter influence only leaving the rest of the parameters to their default values.

3.4. Momentum. Momentum parameter should also be under deep investigation during the design process of each deep learning model. It represents a basic idea of adaptive step during the loss function optimization. For adagrad algorithm it can be written as (6).

$$\theta_{t+1} = \mu\theta_t - \frac{\eta}{\sqrt{\varepsilon I + diag(G_t)}} \cdot g_t \,. \tag{6}$$

Table 3 shows the influence of the μ parameter on the accuracy of the classification done for the presented texture image dataset. Again, for complex textures investigation of this parameter is crucial for accuracy, thus, the system presented herein has the ability to perform such calculations in the batch mode.

Table 3
Momentum parameter influence on tested class accuracy in the system

class	μ				
	0.1	0.5	0.7	0.9	0.99
1	0.629	0.646	0.683	0.914	0.947
2	0.317	0.332	0.45	0.955	0.94
3	0.994	0.996	0.984	0.999	0.997
4	0.988	0.991	0.96	0.986	0.98
5	0.763	0.803	0.823	0.96	0.988
6	0.997	0.999	0.999	0.999	0.999

3.5. Weight decay. Weight decay parameter allows regularization between the system output mean squared function error and a values of model parameters (i.e. network weights θ). This is done in the following way (7).

$$L(\theta) = \frac{1}{N}\sum_{i=1}^{N} L_i(x^i, y^i, \theta) + \lambda R(\theta), \tag{7}$$

where R is the regularization function usually defined as (8).

$$R(\theta) = \frac{1}{N}\sum_{k=1}^{M} \theta_k^2 \,. \tag{8}$$

As can be seen this forms a penalty for high values of the model parameters, thus, providing a solution for a saturated neuron output problem.

Table 4 shows the results obtained for investigated image texture dataset. The same conclusion can be made here, that the value of this parameter is task dependent and shall be investigated for a given problem.

Table 4
Weight decay parameter influence on tested class accuracy in the system

class	λ				
	$5\cdot10^{-6}$	$5\cdot10^{-5}$	$5\cdot10^{-4}$	$5\cdot10^{-3}$	$5\cdot10^{-2}$
1	0.774	0.733	0.719	0.578	0.369
2	0.669	0.692	0.699	0.288	0.181
3	0.999	0.995	0.991	0.961	0.862
4	0.985	0.994	0.989	0.975	0.812
5	0.983	0.87	0.851	0.787	0.694
6	0.999	0.999	0.997	0.993	0.91

Concluding, in the experiment presented above, each texture class was recognized with an accuracy above 90%. That was 93.4%, 96.4%, 99.7%, 99.6%, 99.5% and 99.4% for regular asphalt, asphalt crack, curb, gutters, asphalt contamination, and hatch sewer manhole class respectively. The ROC curves for the entire experiment have been shown in Fig. 6. It can be seen that classes related to the most complex features rise in the lowest rate. All the experiments include the tests to the texture scale and rotation since, as it was mentioned, the system automatically performs data augmentation which, among others, does image rotation and scaling.

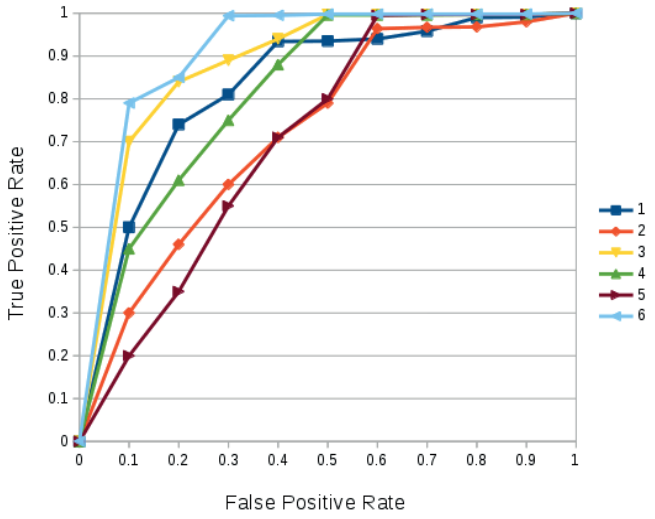


Fig. 6. Receiver Operating Characteristic curves for the asphalt dataset

3.6. Comparison to other solutions and datasets. Previous section described how the training parameters can be fine-tuned in order to achieve best performance in the text recognition task. As an exemplary dataset author used the asphalt dataset with six categories. In order to compare presented solution against other projects other datasets had been investigated. Note, that the approach presented herein relies on pure texture image block that can be categorized to a particular class. Since in the datasets used in this comparison a single image presents a particular texture the segment size in the algorithm presented in this paper had to be set to the size of the input image. This way a single

image produces one segment. The remaining part of the algorithm does not need to be altered.

First experiment was conducted on the STI dataset [18]. It is relatively new dataset that is quite similar in the purpose of its development to the asphalt dataset presented earlier (i.e. it contains different surfaces with some defects for further recognition).

Table 5 presents accuracy results for the tested algorithms in [18] along with the result achieved with DNN approach. As can be seen, due to the transformation to the grayscale image DNN classifier struggles in the task of distinguishing between different colors of travertine texture (columns "Creamy Travertine" and "Orange Travertine"). This leads to the conclusion that introducing the color information to the algorithm could make it more robust to various of textures.

Table 5
Comparison of results achieved for the STI dataset

Texture algorithm	C. Trav.	Hatchet	O. Trav.	avg.
1DLBP	95.6	96.2	95.7	95.8
MLBP16,2	93.6	92.2	94.4	93.4
1DLBP+SSR	97.3	98.0	95.8	97.1
NrCLBP	96.2	97.3	95.9	96.5
MBP	92.3	90.1	91.2	91.3
DNN	82.6	98.1	81.5	87.4

In the next experiments two datasets have been used: MeasTex and Brodatz. Tables 6 and 7 present achieved results. Results for other algorithms were presented in papers [19, 20] for MeasTex and Brodatz dataset respectively.

Table 6
Comparison of results achieved for the MeasTex dataset

Texture algorithm	Grass	Material	OhanDube	VisTex	Avg.
Fractal	90.7	90.8	90.5	81.3	88.3
Gabor	88.5	95.8	98.1	90.1	93.1
GLCM	90.3	95.4	87.4	84.1	89.3
GMRF	92.2	97.1	98.7	92.6	95.2
DNN	97.6	97.4	99.2	93.2	96.9

Table 7
Comparison of results achieved for the Brodatz dataset

Texture algorithm	Avg.
SIFT	91.4
LBP	91.3
WLD	91.2
MLBP	95.1
WLBP	95.7
DNN	98.6

As it can be seen the DNN based algorithm outperformed all the other algorithms in these datasets. Partially it is due to the fact that both datasets originally have grayscale images for all the categories.

4. Conclusions

This paper presents a robust and flexible system for automatic model training and validation for a task of image texture recognition. The system automatically creates an augmented dataset based on simple manual annotations that, next to the image data, are the input for model training. Since the system performs parametrizable data augmentation, it can be used for scaled and rotated textures. In addition, due to the modularized approach of the system architecture particular segments can be replaced or parametrized accordingly to the given task. The biggest novelty however that characterizes the system is the ability to perform analysis of multiple region segmentation for one input texture image. This is because of the fact that the input is analyzed in a sub-window manner where texture recognition occurs. The core of the system is the deep neural network that is automatically configured to match the dimensionality of the input and output. The internal structure of the model was proposed based on the number of experiments performed by the system but can be adjusted since it is stored in an external configuration file.

Experimental results shown in the Section 3 prove that the system can be run in the automatic batch mode in which the influence of the training parameters is investigated. This can be done in a sequential mode on one machine or in a distributed solution since there are no architectural constraints that could disallow it. In the sequential mode, each task is performed back-to-back where the particular parameters are investigated in order to check their influence on the final accuracy. This way the best performing setup can be obtained for a single run and then used in the following experiment. In a distributed approach, the job dispatcher gives a certain task to the calculation nodes and then gathers the data. The influence of the parameters to the model accuracy is then performed by the job dispatcher and this module decides about the end of the experiment procedure.

In addition, the experimental results show that the parameter search investigation performed by the system can lead to extremely good final accuracy values. Thanks to the automatic data augmentation procedure the influence of the image noise to each class can also be investigated in an automatic manner. In this experiment, we left the default setup of the noising module since the main purpose of this paper was to present the system architecture and capabilities, not the texture recognition task itself.

The system also facilitates the so-called cross-validation of the classification accuracy with respect to the training parameters. It can be achieved by validation of the accuracy values across the tables observed for different experiments. This in-

formation is useful for the job dispatcher task and can serve as a clue for further investigation of the training parameters. For example, in Table 2 for class 2 last value is the best performing γ value for this experiment. It can be seen that a similar value (weights of the network are randomly chosen at the beginning of each experiment) of the accuracy is shown in Table 1 for the same class in column 3 ($\eta = 10^{-3}$) since this is the default value of learning rate among all the experiments. Based on these observations the correlation of the training parameters can be investigated as well as the decision of choosing the most efficient parameter values that can be made. Based on Tables 1–3 as well as the ROC curves (Fig. 6) a conclusion can be drawn that some classes are relatively easy for recognition than others (e.g. class 6 (hatch) performs better than class 5 (surface contamination). This happens due to the fact that some classes have the regular texture surface plus the additional features (e.g. surface contamination or crack consists of the regular surface plus the searched object). This fact makes the decision less robust since the detector has to filter out high level of the background first before categorization of the meaningful features.

In summary, the presented system is featured with the ability to train and validate deep neural network models that are then used for robust and accurate texture image segmentation. The trained models can be effectively used in other applications for image texture-based segmentation [21–23] making the presented system a universal tool in the scientific work. It has been proved by performing texture classification experiments on external datasets. Comparison with other state of the art algorithms shows high efficiency of the solution presented herein. In addition, due to the automatic way of problem-solving and parameter tuning the presented system can reduce significantly the design time related to many issues correlated with the training of DNNs [24–26].

The backbone network configuration files as well as the default training setup for caffe library can be downloaded as a prototxt definition files from:

http://rafal.kapela.pracownik.put.poznan.pl/demos.html#.

References

[1] F. Zhou, J.F. Feng, and Q.Y. Shi, "Texture feature based on local Fourier transform", *Proceedings 2001 International Conference on Image Processing*, Thessaloniki, Greece, 2001, vol. 2, pp. 610–613.

[2] P. Wu, Y. Choi, Y. Ro, and C. Won, "MPEG-7 Texture Descriptors", *Int. J. Image Graph.* 1, 547–563, (2001).

[3] R. Kapela, A. Rybarczyk, P. Śniatała, and R. Rudnicki, "Hardware realization of the MPEG-7 Edge Histogram Descriptor", *Mixed Design of Integrated Circuits and Systems, MIXDES*, Gdynia, Poland, 2006, pp. 675–678.

[4] R. Kapela and A. Rybarczyk, "A real-time shape description system based on MPEG-7 descriptors", *J. Syst. Architect.* 53, 602–618 (2007).

[5] A. Abdelhafiz and Y. Mostafa, "Automatic texture mapping mega-projects", *J. Spat. Sci.* 65(3), 467–479 (2020), doi: 10.1080/14498596.2018.1536002.

[6] T. Hermes and A. Miene, "Automatic Texture Classification by Visual Properties", in: *Classification and Information Processing at the Turn of the Millennium. Studies in Classification, Data Analysis, and Knowledge Organization*, R. Decker, W. Gaul (eds), Springer, Berlin, Heidelberg, 2002, doi: 10.1007/978-3-642-57280-7_24.

[7] Lei Qin, Weiqiang Wang, Qingming Huang and Wen Gao, "Unsupervised Texture Classification: Automatically Discover and Classify Texture Patterns", in *18th International Conference on Pattern Recognition (ICPR'06)*, Hong Kong, 2006, pp. 433–436, doi: 10.1109/ICPR.2006.1146.

[8] Y. Guo, G. Zhao, M. Pietikainen, and Z. Xu, "Descriptor Learning Based on Fisher Separation Criterion for Texture Classification", in: *Computer Vision – ACCV 2010. ACCV 2010. Lecture Notes in Computer Science*, R. Kimmel, R. Klette, A. Sugimoto (eds), vol. 6494, pp. 185–198, Springer, Berlin, Heidelberg, 2010, doi: 10.1007/978-3-642-19318-7_15

[9] R.M. Anwer, F.S. Khan,J. van de Weijer, M. Molinier, and J. Laaksonen, "Binary patterns encoded convolutional Neural Netw. for texture recognition and remote sensing scene classification", *ISPRS-J. Photogramm. Remote Sens.* 138, 74–85 (2018), doi: 10.1016/j.isprsjprs.2018.01.023.

[10] S. Basu *et al.*, "Deep neural networks for texture classification – A theoretical analysis", *Neural Netw.* 97, 173–182 (2018), doi: 10.1016/j.neunet.2017.10.001

[11] G. Zhu, B. Li, S. Hong, and B. Mao, "Texture Recognition and Classification Based on Deep Learning", *Sixth International Conference on Advanced Cloud and Big Data (CBD)*, Lanzhou, 2018, pp. 344–348. doi: 10.1109/CBD.2018.00068.

[12] Y. Jia *et al.*, "Caffe: Convolutional architecture for fast feature embedding", arXiv preprint arXiv:1408.5093, 2014.

[13] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier Neural Networks", *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, JMLR Workshop and Conference Proceedings*, PMLR, USA, vol. 15, pp. 315–323. http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf.

[14] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks", *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, PMLR, 2010, vol. 9, pp. 249–256, http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf.

[15] R. Kapela *et al.*, "Asphalt surfaced pavement cracks detection based on histograms of oriented gradients", *22nd International Conference Mixed Design of Integrated Circuits & Systems (MIXDES)*, Torun, 2015, pp. 579–584.

[16] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi, "Describing Textures in the Wild", *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[17] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient sethods for snline learning and stochastic optimization", *J. Mach. Learn. Res.* 12, 2121–2159 (2011).

[18] S.F. Ershad, "A New Benchmark Dataset for Texture Image Analysis and Surface Defect Detection", *CoRR*, arXiv:1906.11561, 2019, doi: 10.13140/RG.2.2.33612.46722.

[19] R. Paget and D. Longstaff, "Nonparametric Markov random field model analysis of the MeasTex test suite", *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*, Barcelona, Spain, 2000, pp. 927–930 vol. 3, doi: 10.1109/ICPR.2000.903696.

[20] F. Liu, Z. Tang, and J. Tang, "WLBP: Weber local binary pattern for local image description", *Neurocomputing* 120, 325–335 (2013).

[21] N Borodinov *et al.*, "Machine learning-based multidomain processing for texture-based image segmentation and analysis", *Appl. Phys. Lett.* 116, 044103 (2020), doi: 10.1063/1.5135328.

[22] D. Koundai, "Texture-based image segmentation using neutrosophic clustering", *IET Image Process.* 11(8), 640–645 (2017), doi: 10.1049/iet-ipr.2017.0046.

[23] T. Markiewicz, Z. Swiderska-Chadaj, J. Gallego, G. Bueno, B. Grala, and M. Lorent, "Deep learning for damaged tissue detection and segmentation in Ki-67 brain tumor specimens based on the U-net model", *Bull. Pol. Ac.: Tech.* 66(6), 849–856 (2018).

[24] T. Poggio and Q. Liao, "Theory I: Deep networks and the curse of dimensionality", *Bull. Pol. Ac.: Tech.* 66(6) 761–773 (2018), doi: 10.24425/bpas.2018.125924.

[25] T. Poggio and Q. Liao, "Theory II: Deep learning and optimization", *Bull. Pol. Ac.: Tech.* 66(6), 775–787 (2018), doi: 10.24425/bpas.2018.125925.

[26] M. Grochowski, A. Kwasigroch, and A. Mikołajczyk, "Selected technical issues of deep Neural Netw. for image classification purposes", *Bull. Pol. Ac.: Tech.* 67(2), 363–376 (2019).